ISI Special Report
ISI/SR-89-235
June 1989

(4)

Alan Katz
Stephen Casner

Supercomputer Workstation Communication

INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/SR-89-235	5. MONITORING ORGANIZATION REPORT NUMBER(S) -------
6a. NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION -------
6c. ADDRESS (City, State, and ZIP Code) 4676 Admiralty Way Marina del Rey, CA 90292-6695	7b. ADDRESS (City, State, and ZIP Code) -------
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-87-C-0719
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. ------- / PROJECT NO. ------- / TASK NO. ------- / WORK UNIT ACCESSION NO. -------

11. TITLE (Include Security Classification)

Supercomputer Workstation Communication (Unclassified)

12. PERSONAL AUTHOR(S)
Katz, Alan; Casner, Stephen

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Special Report	FROM _____ TO _____	1989, June	28

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
computer communication, NeWS, protocols, protocol design, supercomputers, Wideband Satellite Network, X Window System

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is the final report of the Supercomputer Workstation Communication (SWC) project. The project explored new techniques for using high-capacity networks to communicate between personal workstations and remote supercomputers. The effort involved evaluation and testing of standard IP/TCP-based protocols, as well as newer remote window protocols such as MIT's X Window System and Sun Microsystems' NeWS. The DARPA Wideband Satellite Network provided the high-capacity, long-distance communication path for many of these tests. Additional areas of research included a preliminary study of a proposed Intelligent Communication Facility and an investigation of the current and future user requirements of the scientific community that will use supercomputers.

(FP) (—

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Victor Brown Sheila Coyazo	22b. TELEPHONE (Include Area Code) 213/822-1511 / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

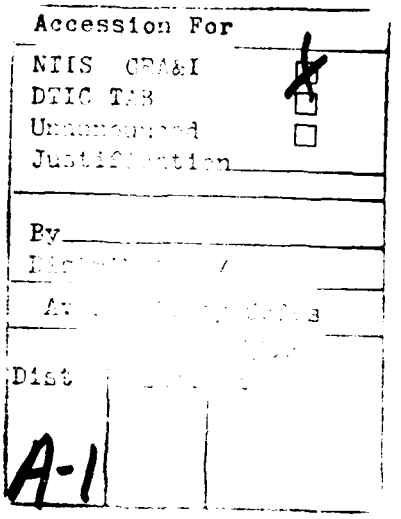
SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Special Report
ISI/SR-89-235
June 1989

University of Southern California

Alan Katz
Stephen Casner

Supercomputer Workstation Communication



INFORMATION SCIENCES INSTITUTE

213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-87-C-0719. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

1 INTRODUCTION

The Supercomputer Workstation Communication (SWC) project has explored new techniques for using high-capacity networks to communicate between personal workstations and remote supercomputers. As part of this effort, the SWC project evaluated and tested the use of standard IP/TCP-based protocols, as well as newer remote window protocols such as the X Window System and Sun Microsystems' NeWS. The DARPA Wideband Satellite Network provided the high-capacity, long-distance communication path for many of these tests. User requirements of the scientific community that will be using these systems were also investigated.

This is the final report for the project, covering the questions we studied, the tests we performed, and the results of our evaluations.

2 PROBLEM BEING SOLVED

The project addressed the need for effective, high-bandwidth communication between powerful remote computers and their users. The need for continued research in this area is demonstrated by the rapid growth of the Defense Data Network (DDN) (and more recently of NSFNET), and especially by the advance of technology that allows higher speed long-haul networks.

Two major initiatives, now under way, are helping to provide significantly increased computing resources to research scientists. The first, a component of the DARPA Strategic Computing Initiative, is working to develop scalable parallel architectures in order to provide a cost-effective form of very high-performance computing. The second, the NSF Supercomputer Initiative, has created a number of supercomputer sites that are connected by a high-capacity backbone network. Each site has its own regional network, which provides access to universities and other research establishments.

With the creation of supercomputer centers and high-capacity networks providing access to them, there will soon be a large number of scientists remotely accessing computer resources in order to do research. For this to be effective, it will be necessary for them to coordinate the use of remote systems with that of local resources such as

mainframe computers, workstations, smaller personal computers, and laboratory equipment. Many technical issues must be resolved for this to work effectively.

The SWC project has attempted to answer questions such as the following:

1. How well do the traditional remote access protocols work on a supercomputer and over a network such as the Wideband Network?

2. Are these protocols adequate, or should additional new protocols be developed?

3. How does one effectively split computer processing between a remote supercomputer and a local workstation? What parts of an application should be run remotely and what parts must be run locally?

4. What is the best way to edit remote files and to interactively debug programs that are running remotely?

5. Is it possible to provide a generally usable means of connecting the output of a program running on one remote machine to the input of a program running on another machine? Are existing transport protocols adequate to handle this?

6. What will the user requirements of the scientific community be? What tools and protocols must exist in order for the supercomputer/workstation environment to be useful?

3 GOALS AND APPROACH

The main objective of the Supercomputer Workstation Communication project was to develop new techniques to allow personal workstations to communicate with remote supercomputers (such as a Cray 2) over high-capacity networks. It is common for supercomputer centers to provide high-bandwidth access for local users over Ethernets, Hyperchannels, or other high-speed networks, but remote users are often constrained by low-data-rate links. The DARPA Wideband Satellite Network, with its 3 Mb/s channel speed, provided the combination of high capacity and long distance we needed for our tests of remote access.

Our research was divided into four areas. The first area of research was the evaluation and testing of existing remote access protocols. We verified that the file

transfer protocol (FTP) and remote login protocol (Telnet) work over the Internet to the five NSF-sponsored supercomputer sites. We also tested the performance of transport protocols over the Wideband Network: the well-established TCP protocol, as part of FTP, and an experimental protocol from MIT called NETBLT.[1]

In our second area of study, we explored the possibility of an Intelligent Communication Facility that would allow users to connect the output of a program running on one machine to the input of a program running on a different machine. It should be possible to "patch together" various programs on different machines, some of which may be supercomputers and some less powerful, without having to modify the programs.

The third area involved research into supercomputer and workstation interaction. We wanted to learn how to share processing between a remote supercomputer and a workstation, how the user should interact with the systems, and how to best use the new remote window protocols. The remote window protocols we studied were MIT's X Window System[2] (X for short) and Sun Microsystems' Network extensible Window System (NeWS).[3]

As part of this effort, we developed a prototype user environment under X and wrote various application programs. One such program was an interactive Mandlebrot Set viewing program. A second application was a unique type of split editor for editing remote files from a personal workstation. This split editor runs within GNU Emacs[4] and includes remote directory viewing, FTP, and remote login capability.

In our last area of investigation, we attempted to predict the future user requirements of the scientific community that will be using workstations to access supercomputers. This effort included a survey of how scientists currently use supercomputers, a study of user interface issues, and a prediction of additional protocols and tools that will be needed for the supercomputer/workstation environment to work effectively.

One of the scientists' requirements is a means to communicate mathematical equations

with one another, but this is hindered by the lack of any standard for representing equations on a computer. We have worked with other groups, including those working on NSF's EXPRES project,[5, 6] to study this area and to establish requirements for such a standard.

4 SCIENTIFIC PROGRESS

The progress made in the Supercomputer Workstation Communication project was in four general areas. Each of these areas will be covered separately.

4.1 Testing and Evaluation of Traditional and New Transport Protocols

Our hypothesis is that effective remote access to the supercomputers will require more sophisticated protocols than the standard file transfer and remote login protocols. However, testing of these protocols established a useful performance baseline. The tests verified the accessibility of the supercomputers over the Internet and the functionality of the protocol implementations on those machines.

4.1.1 Telnet and FTP on the NSF supercomputers

We surveyed the Internet accessibility of the five NSF-established supercomputer centers. Connections were made from a Sun 3 workstation at ISI directly to the supercomputer at each site (where possible) or to a front-end VAX (for the batch-oriented machines). Since none of the supercomputers are connected to the Wideband Satellite Network, we performed these tests over the ARPANET, which was extremely congested at times during the period of our tests (early 1987).

We tested five supercomputers: the Cray XMP running CTSS at the San Diego Supercomputer Center; the Cray XMP running CTSS at the National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign; the IBM 3090 running VM at Cornell University; the Cray XMP running COS at the Pittsburgh Supercomputer Center; and the Cyber 205 running VSOS at the John Von Neumann Center (JVNC) in Princeton. The Cyber 205 was to be replaced by the long-awaited ETA-10, but this was not in place when we did our testing.

Telnet worked to all five sites with only the usual network delay common to the ARPANET at that time. We were able to directly log in to the two CTSS Crays. In order to use Cornell's IBM 3090, it was necessary for us to run a program locally in order to emulate an IBM 3270 intelligent terminal on the Sun. This was the only type of terminal that would work with the 3090.

The last two systems mentioned above are batch-oriented machines and were accessed by logging in to a front-end VAX running the VMS operating system. Jobs were submitted to the supercomputers via a batch queue on the front-end machine. We were able to Telnet to these front-end machines as well.

We were able to FTP files to and from the Cornell, NCSA, and JVNC computers. FTP was not available at San Diego at the time of the tests but was being developed. The Pittsburgh computer had user FTP only, which meant that it was necessary to log in to their system and run FTP in order to transfer files from/to it. This constraint would make it impossible to run a background FTP job from a workstation.[7] Again, this situation may have changed since the time of our test.

It should be mentioned here that UNIX will probably become the operating system for most supercomputers in the future. The JVNC has received its ETA-10 system, which runs UNIX. Also, the Cray sites have plans to change to Unicos, which is UNIX for the Cray.

4.1.2 FTP performance over the Wideband Network

In anticipation of the connection of supercomputers to the Wideband Network, we tested the performance of FTP for file transfers between smaller computers that are already part of the Wideband Network. The performance was limited not by processing power, but by the use of TCP as the transport protocol underneath FTP.

The raw bandwidth of the Wideband Network is relatively high (3 Mb/s), but so is its round-trip delay (almost 2 seconds for datagram traffic). These factors result in a large bandwidth $\times$ delay product; that is, many bits are "in flight" at once. To keep a 1 Mb/s stream flowing would require the transmitter to send 250 kilobytes of data

before waiting for an acknowledgment. For TCP, this number corresponds to the "window size" which, in a typical implementation, is only 2 kilobytes. If the transmitter can send only 2 kilobytes and must then wait 2 seconds for the acknowledgment to be returned from the receiver, the resulting throughput is only 16 Kb/s. Experimental results demonstrated this limitation clearly.

The TCP window size of 2 kilobytes was chosen to avoid overflowing the buffer capacity of gateways when trying to send data through the lower-bandwidth ARPANET. The protocol field carrying the window size is 16 bits, so a window size up to 65,535 bytes could be used without modifying the TCP protocol. Our tests were performed between a Sun workstation at ISI and a VAX at BBN. We were able to adjust the TCP window size in the Sun by patching kernel variables, and we were able to adjust the window size per connection in the BBN VAX TCP implementation through a special command added to the FTP user program.

We found that throughput increased linearly as we increased the window size. At 15 kilobytes, the transfer rate was approximately 60 Kb/s. We were unable to increase the window size to 16 kilobytes or above because of a limitation in the arithmetic of the TCP implementation on the Sun at that time (SunOS 2.0).

To test the capacity of the Wideband Network without acknowledgment delays or window-size limits, we began a separate set of tests using the ICMP Echo Request/Reply protocol as implemented by the "ping" program on a Sun workstation. The ping program transmits a sequence of echo request packets and keeps statistics on the replies that are returned. We were able to transmit 500 Kb/s from ISI to BBN and back to ISI, for a combined 1 Mb/s on the Wideband Network. The packets were 1400 bytes long and were transmitted every 22 milliseconds. In a typical test, the average round-trip time was 1.9 seconds, with a maximum of 2.6 seconds. There was a 0.6 percent packet loss rate on the Wideband Network, and another 0.1 percent of the packets incurred uncorrected bit errors. These tests established the ability of the Wideband Network to carry 1 Mb/s without excessive packet loss and while maintaining expected delay times. Therefore, it seems reasonable to conclude that an

FTP implementation that allowed the full 65,535-byte TCP window size would be able to achieve the calculated 250 Kb/s transfer rate.

Recently, extensions to the TCP protocol have been proposed to provide efficient operation over a path with a high bandwidth × delay product.[8] The window-size limit would be circumvented by negotiating an implicit scale factor to multiply times the window-size value carried in the header field. To reduce the effects of packet loss, another proposed extension would allow selective acknowledgment, so that the receiver could inform the sender about all segments that have arrived successfully; then only the segments actually lost would have to be retransmitted.

The indiscriminate use of large window sizes could cause severe congestion on paths with insufficient capacity. To operate efficiently over a wide range of network performance characteristics, it is necessary for the TCP window size to be adjusted dynamically. A study by Van Jacobson has shown a collection of techniques, including window size adjustment, to avoid congestion.[9]

4.1.3 Tests of NETBLT over the Wideband Network

The NETBLT protocol[1] developed at MIT is a promising alternative that avoids the window-size limitations of TCP. NETBLT is intended for bulk data transfer operations. In cooperation with MIT, we ran several series of tests of NETBLT transfers over the Wideband Network.

In the first series, IBM-PC/ATs were used at ISI and MIT. These machines were located on Ethernet LANs connected by Butterfly gateways to the Wideband Network. The NETBLT traffic was monitored on the Ethernet with the SpyTool program, which was developed at ISI for use in the Xerox Development Environment on a Dandelion Workstation.[10] Throughput was limited by packet loss at the receiving IBM-PC/AT. Bunching of the packets as they flowed through the network caused shorter interpacket intervals than the network interface on the IBM-PC/AT could tolerate.

To avoid the packet loss problem, the NETBLT implementation was ported at ISI to a Sun workstation. The Sun provides increased processing speed and a faster network

interface. After a number of tests in which network timing was analyzed and NETBLT parameters were refined, we were able to achieve transfer rates near the calculated total throughput available to user traffic on the Wideband Network. The overall rate (which includes initial and final handshaking) for a transfer of 716,000 bytes was 521 Kb/s. However, the steady-state transfer rate (once the connection was established) was 942 Kb/s. This compares favorably with the theoretical maximum rate of 1.05 Mb/s calculated from the chosen NETBLT rate-control parameters. A small amount of packet loss at the destination Sun seemed to be the reason for the difference. (These results are reported by Mark Lambert in his report on tests of NETBLT at MIT.[11])

Additional tests were run betweens two Suns, one at BBN and one at ISI, with similar results.

NETBLT performance on the Wideband Network might be improved by operating it over the stream-oriented ST protocol[12, 13] rather than the datagram IP protocol. We have participated in the development and testing of ST for transmission of packet voice and video in the Multimedia Conferencing project. The bandwidth-reservation feature of ST allows for a smooth flow of data at a constant rate across the Wideband Network. This feature would mesh well with the requirements of NETBLT for rate-based flow control. Future work in this area might include the development of an FTP application program using the NETBLT protocol on top of ST.

4.2 Intelligent Communication Facility Study

We did a preliminary study of a system called an Intelligent Communication Facility, which would allow users to connect the output of a program running on one machine to the input of a program running on a different machine. It would be very valuable if programs on different machines, some of which may be supercomputers and some less powerful, could be linked in this way without the need for modifying the programs.

One way to accomplish this might be to extend the idea of UNIX pipes to contain typed data. One could think of these ***typed pipes*** as carrying a stream of one type of data from a process on one machine to another process on a different machine. Since

the connection would be like a UNIX pipe, there would be no direct indication of how much data would be coming down it. Therefore, in order to make this type of connection work with remote machines, a remote execution protocol similar to UNIX's rsh must also exist. (We have found that such a protocol is necessary for many other applications and will discuss this further in Section 4.3.2 of this report).

In order to comply with international standards, we suggest that the data in these pipes conform to the CCITT X.409 data representation standard.[14] (See also "A Survey of Data Representation Standards."[15]) The X.409 standard originally did not include a representation for floating-point numbers, essential for scientific applications, but it has since been modified to include them. The pipes themselves could be implemented on top of TCP.

In a typical application, there may be a program that does a large amount of computation (say, running on a supercomputer) whose output consists of many pages of numbers. Another program on a second machine might accept numerical input and produce graphical data, which the user would like to display on yet a third machine, a workstation. These programs may have been written by different people. The sources to them may be unavailable. Therefore, it is desirable to have a way for these programs to communicate with one another under the direction of a user at one of the machines, without the programs themselves having to be modified in any way.

We suggest that a library of ***adapters*** should be created. Each adapter would be a small program that would read in data in a given format and then output an X.409 stream. Suppose, for example, that the first program's output consisted of a table with each line consisting of six floating-point numbers (say, in FORTRAN format 6F10.6). The adapter would read in such a line, ignore any header or title information, and output a list of X.409 floating-point numbers. On the receiving side, the second program would have an inverse adapter that would read in the X.409 list of floating-point numbers and output the equivalent data in FORTRAN format. These types of connections would continue to the third machine, the workstation (see Figure 1).

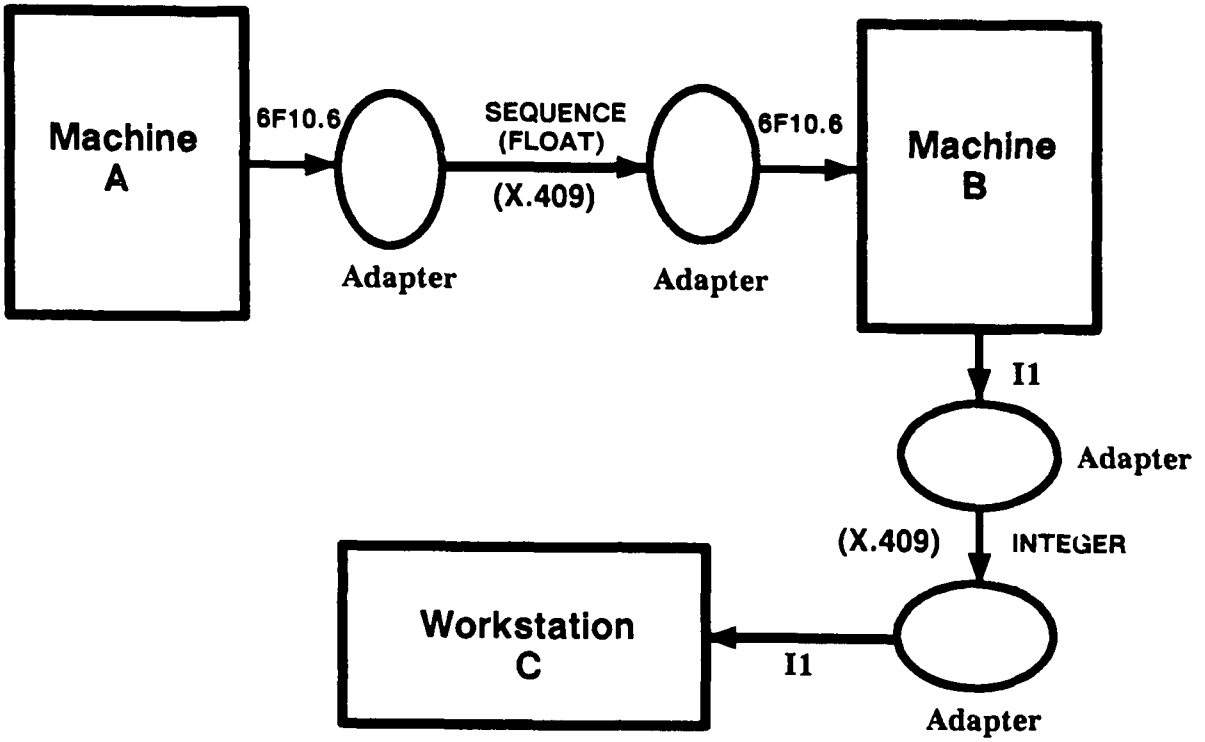


Figure 1: Interconnection of typed pipes across machines

The connections to the adapters, as well as the network typed-pipe connections, could be specified by the user logged in to any of the three machines. This could be as simple as the way it is done in UNIX (typing the names of the programs to be executed, with special characters to denote connections to pipes), or it could be much more sophisticated. One possibility is a graphical interface on the workstation that would allow a user to browse a database of programs and adapters and to specify the interconnections. Something like this was studied by Brown.[16]

If a program were written to allow the input of multiple data sets, the above system could allow for multiple pipes of input from different processes to be connected to it. Also, this type of system could work well with the remote window protocols described in the next section, allowing the older non-interactive programs to be used with a front-end that knew about the MIT X Window protocol or Sun's NeWS. The output of one such program could be in a format like PostScript on top of the X.409 connection. The user could then connect this pipe either to a display process on a workstation or directly to a PostScript printer.

We feel the constraint that the programs need not be modified is important. A general-purpose adapter capable enough to figure out the format of any type of output data is probably beyond current capabilities. However in most cases, a user would know what type of data a program generates and could select a standard adapter or write his own. More work in this area is required.

4.3 Supercomputer/Workstation Interaction and Remote Window Protocols

The bulk of our project's research effort has been in the area of supercomputer/workstation interaction. This involves understanding how to split processing between the remote supercomputer and the local workstation. It is desirable for a user to be able to do this without having to devise a new communications protocol between local and remote processes for each new application.

The new remote window protocols such as the MIT X Window System allow programs running on a remote computer to display windows and to interact with the user on a local workstation in a machine-independent way. This is one natural way to split the processing between the workstation and a supercomputer. We would like to understand when it is useful to do this and when it may be better to have more of the computation occur on the workstation.

4.3.1 The spectrum of supercomputer/workstation interaction

In order to gain a clearer idea of this concept, it is important to keep in mind what a user needs in order to run remote programs. First, there is a need for some method of editing the programs and creating the data that will run remotely. Next, there must be some way of remotely executing these programs. The user will probably need some type of interactive debugger; in addition, since supercomputers generally use a pipelined architecture, an interactive method of measuring performance and of vectorizing code may be necessary. Finally, there must be some method of displaying the results of the computation. The user also might need to interactively modify the course of the remote computation based on a current display of its progress (for example, in a large simulation).

It is useful to consider a spectrum of supercomputer/workstation interaction (see Figure 2). At one end of the spectrum (leftmost on the diagram), as much processing as possible is done on the remote supercomputer, and a minimal amount is done locally. At the other end (at the right), the supercomputer generates only data, and all display and interaction with the user is done by the workstation. Between these two extremes, interaction with the user is shared between the workstation and the remote supercomputer. Below the spectrum line in the figure, some typical types of applications are located in their appropriate places in the spectrum.

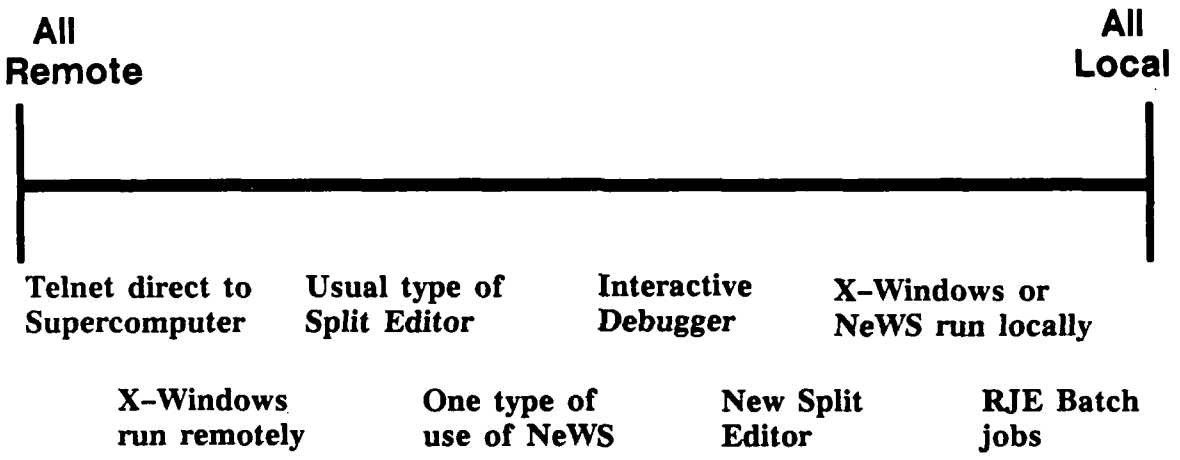


Figure 2: The spectrum of supercomputer/workstation interaction

We should first define the two ends of this spectrum more clearly. In order to do this, let us distinguish between the interactive part of an application (displaying graphical data in a window, interacting with the user via mouse and keyboard, etc.) and the non-interactive part. The non-interactive pieces can be run in a batch mode; that is, they (possibly) read in an input data set, do some (possibly a very large amount of) computation, and create an output data set. The user does not need to interact with this part of the program in order for it to complete its computation.

In order to take advantage of the high processing speed of the supercomputer, the user will want to run at least the non-interactive portion of an application remotely. Also, supercomputer-specific parts (such as computations that take advantage of the pipelined

architecture and the execution part of a vectorizer and debugger) must be run on the supercomputer. This defines the all-local edge of the spectrum. (If we were to go any further, we would not use the supercomputer at all!)

Direct interaction with the user (paying attention to mouse clicks and characters typed) must be done on the workstation. For example, most if not all of an interactive editor should probably run on the workstation, not the supercomputer. In addition, any display of graphical information must be done locally. This defines the all-remote edge of the spectrum. Telneting directly to the supercomputer would use the workstation only to interact with the keyboard and to display text. Since this mode of operation would not allow the display of graphical information and would not take advantage of the workstation's capabilities, we will not consider it further.

A more useful approach, near the all-remote edge of the spectrum, is to use a remote window system such as the MIT X Window System[2] or Sun's NeWS.[3] In this type of system, the workstation acts as a server to process keyboard and mouse activity and to display bitmap data, but *all* other computation occurs in the remote computer. Because the specification of a display in the NeWS environment is a PostScript program, one possible use of this system would be to download some of the display processing to the workstation as PostScript (this is the "One type of use of NeWS" in the middle of Figure 2).

The usual types of ***split editors***, which allow the editing of remote files on a local computer, are also shown in Figure 2. As an experiment, we have developed a split editor (described in Section 4.3.4) that is located more toward the local end of the spectrum.

Generally, if one chooses to work in the middle of the interaction spectrum, it will be necessary to define a communications protocol to specify how the remote and local processes will interact. A future area of research would be to determine how to allow users to do this without having to completely design such a protocol. NeWS provides some of this capability (via PostScript).

At the all-local edge of the spectrum, a user may edit a program entirely at the workstation, submit it as a remote batch job to the supercomputer, transfer the output data back to the workstation, and do all display and interaction with the data at the workstation. For many applications, this is a useful mode of operation. Another project at ISI has developed an interface to FTP that works in the background to support this mode of operation.[7]

This batch-oriented style requires a generally available, IP-based Remote Job Entry (RJE) protocol. Although a general RJE protocol does not yet exist within the Internet, there are systems that work within certain segments of the Internet community. For example, an RJE system is availible on BITNET, but one must be a BITNET host to use it. The National Center for Atmospheric Research (NCAR) has developed a very nice Internet RJE system based on FTP[17] for providing remote access to their supercomputers.

4.3.2 Remote window protocols, the X Window System, and NeWS

Originally, we proposed to develop both a Bitmap-Telnet protocol and a remote window protocol as part of this project. However, by the time the project was started, two such protocols had been developed and are now becoming standards (MIT's X Window system and Sun's NeWS). We therefore felt it was inappropriate to create yet another protocol, and we concentrated instead on how to best use X or NeWS with supercomputers.

Both X and NeWS use a client-server model of window interaction. The user's workstation is viewed as a resource (a bitmap display, mouse, and keyboard) that is allocated by a server program running on the workstation. The server processes requests from client application programs, which may be running either on the workstation or remotely on another machine.

The major technical difference between the two protocols is that the display in a window in X is a raw bitmap and the display in a NeWS window is in PostScript format (a printer description language). Thus, a NeWS display is both hardware and

resolution independent. An X display is hardware independent, but resolution dependent.

X currently seems to be in far wider use than NeWS, probably because it is available essentially free from MIT, while NeWS is a commercial product. However, Sun's merged X/NeWS window package is planned to be a standard part of UNIX, so NeWS may catch on more in the future.

X will be offered as a part of the Unicos Cray operating system and also should be a part of the next release of IBM's VM, the operating system of Cornell's supercomputer. Thus, we expect that many supercomputer users will use at least X, if not both X and NeWS.

Most current and proposed applications that use X or NeWS use these systems because they are rapidly becoming a window protocol standard, not because of their remote capability. Additional protocols are needed for these systems to be used remotely.

Primarily, a remote execution protocol is necessary. If a user on a workstation wants to access a remote supercomputer with X, something must create a process on the remote system; only then can X display this process on the workstation and allow the user to interact with it. The rsh facility in UNIX works as a remote execution protocol, but it has problems with security and authentication. Also, both the workstation and the remote computer must be running UNIX. There is a real need for a general remote execution protocol that will accept a user's password and start up a program (similar to the way FTP works).

Remote applications must be designed carefully in order to minimize their required network bandwidth. It is quite possible in either of these systems to create a large number of packets in a simple application.

4.3.3 Experimentation and testing with X

Some experimentation and testing was done with the X Window System. All work was done with Version 10, Release 4, on a Sun 3 workstation. The standard version of the X protocol, Version 11, has since been released by MIT.

We began with a small demonstration program, running on a VAX over our 10 Mb/s Ethernet, which continually drew moving balls in a Sun window. Using the ISI SpyTool described in Section 4.1.3, we found that 50 packets per second were sent to the Sun and the same number were sent back to the VAX (presumably acknowledgments). The exact same number was generated running multiple programs in different windows. This indicates that the local X server on the Sun was the limiting factor and that this should be typical of the maximum traffic one workstation might generate using X remotely. In actual practice, remote applications should be designed to exchange far fewer packets.

In addition to this testing, we built a number of applications under X. We created a package of small files that would allow a naive user of X to try it. This package, called MyX, creates an environment similar to Suntools (the Sun window package) as far as mouse interaction and window management is concerned. MyX provides users a way to try the X environment without having to wade through a large amount of documentation. We distributed MyX to a number of people throughout the Internet.

We developed an X-based graphics interface for GNU Emacs.* Our graphics extension allows LISP functions within this editor to create simple line-drawing graphics in an X window on the user's workstation.

We also created a larger application under X: a system for interactively viewing Mandlebrot set fractals. It allows the user to "zoom in" on a smaller region of the display. The Mandlebrot set data is calculated as it is displayed. This program runs on a remote VAX and is written in C using the Xlib interface to X.[18] If X had been

---

*At the time of this development, GNU Emacs was one of the few editors that were able to use the X protocol.

available on a supercomputer, it would have been interesting to run this program on one, as examining small areas of the Mandlebrot set can take quite a large amount of computation.

This application is a example of something near the all-remote edge of the supercomputer/workstation interaction spectrum. We would have liked to test a version closer to the other end of that spectrum, but we did not have the time to do so. Such a program might have the remote computer generate a large amount of data into a file; then a local display program would access parts of that file depending on the view the user selected.

Most of the experimental programs we developed under X were done without the use of a toolkit. However, we did write some small programs with DEC's Xt toolkit.[19] In Version 10 of X, a toolkit was a convenience but was not necessary; however, in version 11, use of some kind of toolkit is almost essential. Selection of a toolkit and a window manager (a special client program, run locally, which allows resizing and movement of windows) is a major decision a program developer must make; it will have great influence on the application's development.

We found that using a toolkit allows programming at a somewhat higher level and that programs can generally be made much shorter (as demonstrated by Rosenthal[20]). However, the existing toolkits still require programming at a relatively low level and seem more directed at systems programmers then at general application developers. There is a need for a somewhat higher level toolkit on top of the existing ones. (Note: as things are changing very rapidly, this level of toolkit may already be available. CMU's Andrew toolkit, which we did not have a chance to evaluate due to time constraints, appears to be the type of toolkit we describe.)

4.3.4 A new kind of remote split editor running within GNU Emacs

After considering the various types of applications on the supercomputer/workstation interaction spectrum of Figure 2, we felt it would be useful to prototype an application that falls nearer to the middle. We chose the problem of editing remote files as an example.

One usual way to edit a remote file is to FTP the file to the local workstation, edit it, and then FTP it back to the remote site. Another is to run an editor remotely on top of an interactive network connection (e.g., Telnet). Now, a user can take advantage of X or NeWS to run the editor remotely and display on the local workstation.

Both of the above approaches have potential problems. Editing a program, running it remotely, changing it, and then running it again involves transferring the entire file back and forth across the network, wasting network bandwidth. On the other hand, round-trip network delay time can make interactive remote editing very difficult.

One solution to these problems is a ***split editor***, which divides the editing computation between the local and remote machines. (An example of this is the SED editor, developed for MFEnet users.[21] Another example, closer in design to our prototype, is described by Comer, et al.[22]) Typically, a split editor works by sending one section of the file at a time to the remote machine. As the file is modified, the new versions of these sections are sent back to the remote machine. The user may run into trouble, however, when moving to a part of the file that has not yet been loaded into the local machine. When this happens, the user suddenly must wait until that section is transferred, which can be very frustrating when response has previously been very fast.

In our prototype editor, we assumed that, primarily, the user would be continually making changes to an existing remote file. If the user is creating a large file from scratch, he or she would be more likely to use a standard editor to create the file locally and then transfer the entire file to the remote machine. In our editor, we transfer the entire remote file to the local machine only once at the beginning of the editing session. From then on, we send only the changes that are made to it. The remote side of the editor makes these changes as it receives them and then acknowledges that it has done so to the local editing process. (A similar approach is also taken in a system at Purdue University.[22])

GNU Emacs[4] is an extensible editor that runs its own LISP interpreter. Editing functions are programmed in LISP and can be bound to any key. Instead of writing an

editor entirely from scratch, we decided to build the split editor on top of GNU Emacs. The entire system is written in GNU Emacs LISP and requires no modification of the compiled C code that underlies the LISP. Use of the split editor appears virtually identical to the standard way in which Emacs is used.

When the user selects a remote file to edit, the entire file is transferred to the local machine and the remote side of the editor is started. Nothing is transferred to the remote side until the user makes an actual modification to the file. Then this change is sent to the remote process, which acknowledges it. The remote version of the file is saved periodically in case either machine goes down.

Changes to the file are simply GNU Emacs LISP instructions. If these instructions were saved in a file, and that LISP file were loaded into Emacs, the resulting instructions would convert the old version of the edited file to the new version. Thus, it is also possible to keep a running log of changes and to execute them all at once instead of incrementally.

Our split editor worked well. However, as was the case with the remote window protocols, a remote execution procedure was again necessary. For this split editor, we used the rsh facility of UNIX. Thus, the system as currently implemented requires both the local and remote sites to run UNIX and to have rsh access. Again, this points out the need for a general remote execution protocol not tied to any particular operating system.

As mentioned in Section 4.3.1, when one has an application in the middle of the interaction spectrum one generally must define a protocol for how the remote and local sides will communicate. In the case of our split editor, our protocol is just LISP instructions. This was done for ease of debugging and for readability of the changes, and because it allows for a very simple remote side. The remote process is simply another GNU Emacs process that reads in LISP instructions and executes them.

In addition to its basic editing functions, the split editor allows the user to view

remote files without modifying them and to manipulate remote directories in a way similar to Emacs' Dired mode.[4]

4.4 Future User Requirements of the Scientific Community

In order to maximize the usefulness of supercomputers to the scientific community, it is important to determine the ways in which this community expects to use them. We therefore investigated how researchers currently use supercomputers, how they expect to interact with them in the future, and what protocols and capabilities will be necessary to support this.

We participated in the DARPA Internet Task Force on Scientific Requirements to explore these issues and to make known our results. In addition to releasing a variety of position papers on future scientific requirements for networking, the task force contributed to the 1987 FCCSET report to Congress.[23]

We interviewed several scientists who use supercomputers in order to learn what facilities currently exist and what new ones are desired. We also participated in supercomputer workshops held at NSF-sponsored supercomputer centers.

Because of the importance of mathematical equations in today's scientific communications and the lack of any standard for representing equations on a computer, we also studied the issues involved in defining an equations representation standard. Some user-interface considerations were also studied. The results were reported by Katz[24] and in a soon-to-be-released position paper by the Internet Task Force on Scientific Requirements.[25] We also participated in discussions and meetings with other groups interested in this subject, including those funded under the NSF EXPRES project (some of which was reported by Arnon[26]).

We believe there is a need for a standard that will allow for the interchange of equations between electronic mail messages, various document preparation systems, and symbolic mathematics systems. We have identified four levels of abstraction at which mathematical expressions may be represented. At the present time, there is general

disagreement about which level or levels would be most appropriate for a standard. It is probable that more than one of these levels will be required. However, we believe the development of such a standard will be critical for effective computer-based communication in the sciences.

5 IMPACT

As a result of the NSF Supercomputer Initiative and the DARPA Strategic Computing Initiative, and because of continued increases in network and computer capability, the bulk of the scientific community will soon be using computers interconnected via the Internet in order to do their work. Many will use remote supercomputers; others will use these resources mainly to communicate electronically with their colleagues. As a result of our work, we hope they will be able to work in this new environment more effectively.

Our testing of new and existing protocols will help to determine which protocols should be used in the supercomputer/workstation environment. The development of a system similar to our proposed Intelligent Communication Facility will allow users to run existing programs on different machines without having to rewrite them or to become knowledgeable about low-level network protocols.

The current trend away from the use of large, timeshared mainframes and toward the use of powerful workstations connected to high-speed networks indicates that more and more researchers will use the new remote window protocols such as X and NeWS. Our studies in this area have given examples of how to best use these protocols and have identified several major issues that must be resolved in order for them to work.

The spectrum of supercomputer/workstation interaction provides a method for classifying different modes of using remote computing resources.

It is hoped that our work on an equations representation standard will influence the much-needed work in this area. Such a standard will be important to scientists wishing to collaborate electronically. Through our participation in various workshops and in

the Internet Task Force on Scientific Requirements, we have worked toward the development of such collaborative scientific work.

6 FUTURE WORK

Although the Supercomputer Workstation Communication project is at an end, there is ample opportunity for follow-on work. Continuing research is needed in the following areas:

1. A general Internet remote execution protocol must be developed. This protocol should allow for user authentication and for security. Such a protocol is needed for the remote window protocols to be used effectively, as well as for applications such as our split editor and the Intelligent Communication Facility.

2. An Internet-based batch protocol is also needed. The National Center for Atmospheric Research (NCAR) has developed a such a system on top of existing protocols,[17] but this system requires the user to register a password for his local machine with the batch processor. A separate protocol is really needed, perhaps as a part of a general remote execution protocol.

3. Much more work should be done to test and tune both X and NeWS on supercomputers over high-capacity networks. At the time of our study, neither X nor NeWS was available on any of the NSF-sponsored supercomputers, but they should become available in the near future. It is unknown how much the widespread use of these protocols will load the supercomputers and the networks.

4. A standard must be defined to allow for the interchange of equations among electronic mail messages, various document preparation systems, and symbolic mathematics systems. As more and more scientists become connected to the Internet, they will demand the ability to send mathematical expressions via electronic mail.[24, 25] It would be useful to prototype the Intelligent Communication Facility described in Section 4.2. Having such a system in addition to X or NeWS would allow more effective use of both remote and local resources.

7 REFERENCES

1. Clark, D., M. Lambert, and L. Zhang, "NETBLT: A Bulk Data Transfer Protocol", RFC 998, MIT Laboratory for Computer Science, March 1987.

2. Scheifler, R., and J. Gettys, "The X Window System", *ACM Transactions on Graphics, Special Issue on User Interface Software*, No. 63, 1986.

3. Sun Microsystems, Inc., *NeWS Preliminary Technical Overview,* October 1986.

4. Stallman, R., *GNU Emacs Manual,* Free Software Foundation, March 1987.

5. CSNET, "NSF initiates EXPRES program", *CSNET News*, Vol. 12, Winter 1986.

6. CSNET, "EXPRES Project completes first year", *CSNET News*, Vol. 16, Winter 1988.

7. DeSchon, A., and R. Braden, "Background File Transfer Program (BFTP)", RFC 1068, USC/Information Sciences Institute, August 1988.

8. Jacobson, V., and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, USC/Information Sciences Institute, October 1988.

9. Jacobson, V., "Congestion avoidance and control", *Computer Communications Review,* Vol. 18, No. 4, August 1988, (SIGCOMM '88 Symposium.)

10. DeSchon, A., "USC Information Sciences Institute, Xerox University grant program activities", *Proceedings of the Xerox College and University Computer Research Communications Meeting*, Xerox Corporation, Leesburg, Virginia, November 1986.

11. Lambert, M., "On Testing the NETBLT Protocol over Diverse Networks", RFC 1030, MIT Laboratory for Computer Science, November 1987.

12. Forgie, J. W., "ST - A Proposed Internet Stream Protocol", IEN 119, MIT Lincoln Laboratory, September 1979.

13. Topolcic, C., and P. Park, "Proposed Changes to the Experimental Internet Stream Protocol (ST)", Tech. report, Bolt Beranek and Newman, Inc., April 1987.

14. CCITT, *Message Handling Systems: Presentation Transfer Syntax and Notation, Recommendation X.409, Document AP VIII-66-E,* International Telegraph and Telephone Consultative Committee (CCITT), Malaga-Torremolinos, 1984.

15. DeSchon, A., "A Survey of Data Representation Standards", RFC 971, USC/Information Sciences Institute, January 1986.

16. Brown, R. L., *A Distributed Program Composition System,* PhD dissertation, Purdue University, May 1988.

17. Bassett, B., "NCAR Internet remote job entry system", *Fourth Annual Workshop on Networking and Supercomputers*, Boulder, Colorado, June 1988.

18. Gettys, J., R. Newman, and T. Fera, "Xlib-C Language X Interface, Protocol Version 10", Tech. report, MIT Project Athena, November 1986.

19. Digital Equipment Corporation, *The Xt Toolkit*, 1987.

20. Rosenthal, D., "A Simple X.11 Client Program, or How Hard Can it Really Be to Write 'Hello, World'?", Tech. report, Sun Microsystems, Inc., October 1987.

21. Maron, N., "Using a personal computer as a workstation extension to the Cray/CTSS mainframe environment", *Proceedings of a Workshop on Supercomputing Environments*, NASA Ames Research Centler, Moffett Field, California, June 1986.

22. Comer, D., J. Griffioen, and R. Yavatkar, "Shadow editing: A distributed service for supercomputer access", *Proceedings of the 8th International Conference on Distributed Computer Systems*, The Computer Society and IEEE, San Jose, California, June 1988.

23. Federal Coordinating Council on Science, Engineering, and Technology, "A Report to the Congress on Computer Networks to Support Research in the United States", Tech. report, FCCSET, June 1987.

24. Katz, A., "Issues in Defining an Equation Representation Standard", RFC 1003, USC/Information Sciences Institute, March 1987, Also published in *ACM SIGSAM Bulletin*, May 1987.

25. Katz, A., "Towards a Standard for the Representation of Mathematical Expressions", position paper, Internet Task Force on Scientific Requirements, 1989, (To be released.)

26. Arnon, D., "Report of the workshop on environments for computational mathematics", *SIGGRAPH Conference*, ACM, Anaheim, California, July 1987.